UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Cómo usar un millón de imágenes para etiquetarme una

Sergio Gómez Nieto

JULIO 2014

RESUMEN.

Muchas imágenes, vienen acompañadas de gran cantidad de texto que se relaciona con dicha imagen de algún modo. Dentro de esto texto podemos encontrar palabras que pueden describir la imagen, que sirven como etiqueta o "tag" para definirla. El problema que se nos plantea es como obtener esas palabras y decidir cuál describe mejor la imagen.

El objetivo que nos hemos propuesto en este proyecto es desarrollar una herramienta para a obtener de un texto las palabras que mejor describen una imagen.

Primero plantearemos un algoritmo para extraer las palabras más importantes de un texto para posteriormente decidir que palabra es la que mejor describe la imagen. Para ello realizaremos búsquedas sobre una base de datos de imágenes de esas palabras para posteriormente realizar una comparación de esas imágenes con la que queremos etiquetar.

La comparación de imágenes se basará en comparar los histogramas de color y la comparación de bordes. Ya entraremos en detalle en cuerpo del documento, pero al final obtendremos un resultado numérico que medirá el grado de similitud de ambas imágenes.

Cuando tengamos los resultados de comparar todas las imágenes de cada palabra podremos decidir, aplicando algún método (como por ejemplo la media aritmética) que palabra produce los mejores resultados.

Desarrollaremos este algoritmo en una serie de bibliotecas que implementan cada módulo del algoritmo (selección de palabras, búsqueda de imágenes, comparación de imágenes y etiquetado de la imagen) y probaremos los resultados con un programa de prueba.

En el presente documento se detallará en primer lugar el funcionamiento del algoritmo y algunos aspectos teóricos en los que se basa. A continuación, se explican las tecnologías utilizadas y por qué hemos optado por ellas. Luego se mostrará el diseño de las bibliotecas desarrolladas. Finalmente, enseñaremos un ejemplo de ejecución en el que se etiquetará una imagen.

- **Palabras clave:** búsqueda de imágenes, comparación de imágenes, imagen, etiqueta, Búsqueda y Recuperación de Información, OpenCV, Bing, C#.

SUMMARY.

Many images are associated to plenty of text that describe the image somehow. Within this text, you can find words that can describe the image, which you can use as a label or "tag" to define it. The problem we face is how to get those words and decide which describes the picture best.

The goal we have proposed in this project is to develop a tool to obtain the words in a text that best describe a picture.

First we created an algorithm to extract the most important words in a text to later decide the best word that describes the picture. For that, we make queries on an images database with those words to make a comparison between those images and the one we want to tag.

Image comparison is based on comparing the color histograms and comparing edges. We go into detail later in the document, but in the end we get a numerical result that measure the degree of similarity of two images.

When we compare the results of all images of every word we decide, using some method (such as the arithmetic mean) the word that produces the best results.

We develop this algorithm in a number of libraries that implement each algorithm module (word choice, image search, image comparison and tagging the image) and a test program.

This paper will detail the performance of the algorithm and some theoretical aspects on which it is based. Then we explain the technologies used are and why we chose them. After this, we show the design of the library we developed. Finally, we show the results of the test program.

- **Key Words:** image search, image comparison, image, tag, information retrieval, OpenCV, Bing, C#.

GLOSARIO.

- **Algoritmo:** un algoritmo es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad.
- **Módulo:** es una parte repetitiva, autónoma e intercambiable de un diseño modular.
- **Etiqueta (tag):** es una palabra clave asignada a un dato almacenado en un repositorio. Proporcionan información que describe el dato.
- **Base de datos:** es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.
- **Longlist:** archivo de palabras comunes.
- **Stemming:** es un método para reducir una palabra a su raíz o (en inglés) a un stem o lema.
- **Stem:** raíz o lema de una palabra.
- **ShortCorpus:** archivo con todas las palabras que aparecen en la versión corta del British National Corpus con el número de veces que aparece.
- **Histograma:** en estadística, un histograma es una representación gráfica de una variable en forma de barras.
- **Píxel:** es la menor unidad homogénea en color que forma parte de una imagen digital.
- **Matriz:** es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo. Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).
- **Media:** la media de un conjunto finito de números es el valor característico de una serie de datos cuantitativos objeto de estudio que parte del principio de la esperanza matemática o valor esperado, se obtiene a partir de la suma de todos sus valores dividida entre el número de sumandos.
- **Desviación:** es una medida de dispersión para variables de razón (variables cuantitativas o cantidades racionales) y de intervalo. Se define como la raíz cuadrada de la varianza de la variable.
- **API:** interfaz de programación de aplicaciones (IPA) o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- **Motor de búsqueda:** es un sistema informático que busca archivos almacenados en servidores web.
- **Biblioteca:** (del inglés library) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- **Clase:** es una plantilla para la creación de objetos de datos según un modelo predefinido. Son un pilar fundamental de la programación orientada a objetos. Cada objeto creado a partir de la clase se denomina instancia de la clase.
- **Entorno de desarrollo integrado:** llamado también IDE (integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación.
- **Programación orientada a objetos:** es un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos.
- **Aplicación:** es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos.
- **Constructor:** es una subrutina cuya misión es inicializar un objeto de una clase. En el constructor se asignan los valores iniciales del nuevo objeto.
- **Diagrama de clases:** es un tipo de esquema que describe la estructura de un sistema mediante sus clases.
- **Identificador de recursos uniforme** o **URI:** en inglés Uniform Resource Identifier, es una cadena de caracteres que identifica los recursos de una red de forma unívoca.
- **Localizador de recursos uniforme** o **URL**: en inglés de Uniform Resource Locator, es un identificador de recursos uniforme (URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo.

ÍNDICE DE CONTENIDO.

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

1. INTRODUCCIÓN.

1.1. Motivación.

Muchas imágenes, sobre todo en la red, vienen asociadas a una buena cantidad de texto, en los que se "esconden" unos términos que describen muy bien el contenido de la imagen. El problema es cómo identificar y aislar estas palabras entre todas las palabras del texto.

Las técnicas de *information retrieval* [1] nos ayudan algo, en el sentido que nos ayudan a identificar las palabras que mejor describen el contenido del documento. Pero, ¿Cuáles de estas palabras describen el contenido de la imagen? En este proyecto experimentaremos con una idea muy sencilla para asociar etiquetas (*tags*) a este tipo de imágenes.

Digamos que tenemos una gran base de datos con imágenes y texto, y que, una vez aisladas las palabras que mejor describen el documento, las usamos para lanzar consultas en esta base de datos. La idea con que trabajaremos es que las palabras que describen la imagen conseguirán resultados muy parecidos a la imagen que estamos analizando, mientras las palabras que describen el documento pero no la imagen conseguirán resultados muy diferentes.

Por ejemplo, si tenemos una imagen de Obama en un documento que contiene los términos "Obama" y "White House", haremos una consulta a la base de datos con estos dos términos. El primero nos dará muchas imágenes de Obama (parecidas a la que tenemos), mientras que el segundo nos dará muchas imágenes de la Casa Blanca (distintas de la que tenemos). Así analizando la similitud entre la imagen que tenemos y los resultados de la base de datos, podemos descubrir cuáles son las palabras que mejor describen el contenido de nuestra imagen.

1.2. Objetivos.

El objetivo de este trabajo es desarrollar un programa que recibiendo una imagen y un texto asociado a esa imagen, seleccione la palabra del texto que mejor describa la imagen.

1.3. Estructura del documento.

El presente documento está divido en seis capítulos principales (siendo el primero de ellos esta introducción).

El **capítulo 2** explica el algoritmo de la aplicación, los pasos que se deben dar para la resolución del problema planteado.

En el **capítulo 3** se exponen las distintas tecnologías utilizadas en el desarrollo de la aplicación.

En el **capítulo 4**, en primer lugar se presenta la arquitectura de la aplicación, los módulos que la componen y la comunicación entre ellos, para posteriormente explicar de forma detallada cada uno de estos módulos.

En el **capítulo 5** se expone el funcionamiento de la aplicación paso a paso a través de un ejemplo práctico.

Para finalizar, en el **capítulo 6** se presentan las conclusiones de este trabajo, destacando las posibles mejoras que se pueden realizar en el sistema desarrollado y sus posibles aplicaciones.

2. ALGORITMO.

El objetivo de la aplicación, como se ha comentado anteriormente, es obtener una palabra a partir de un texto pasa asociar a una imagen. Para ello dividimos el problema en cuatro partes:

- Selección de las palabras más relevantes del texto.
- Realizar búsquedas sobre la base de datos de las palabras seleccionadas.
- Comparación de las imágenes obtenidas con nuestra imagen.
- Decidir la etiqueta (procedimiento principal).

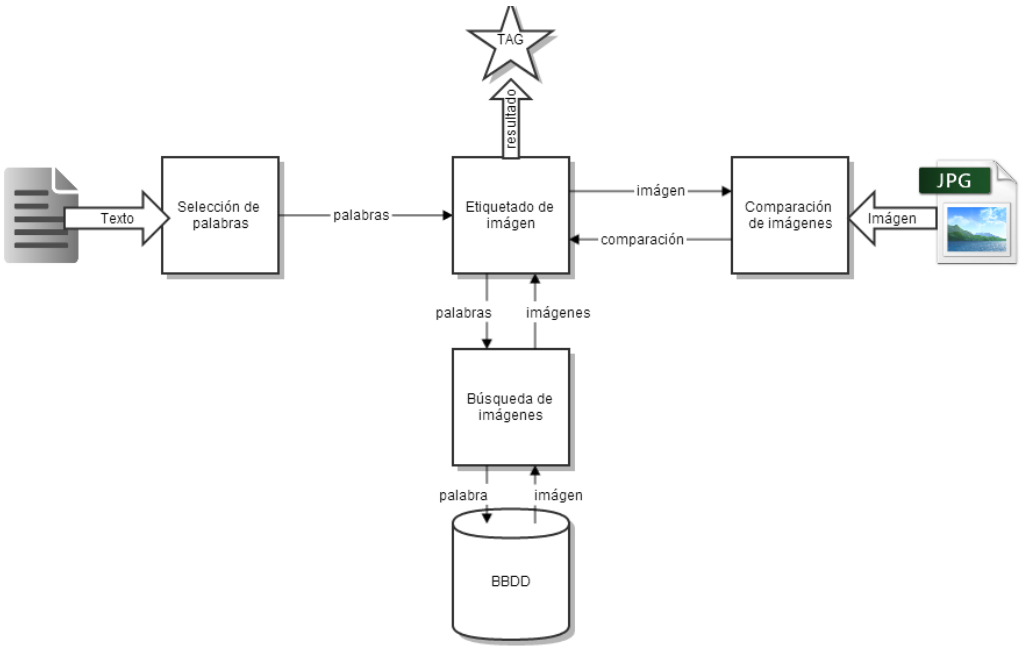

Figura 1: Esquema del algoritmo

Primero se seleccionan las palabras más relevantes del texto. A continuación se realiza una búsqueda de imágenes en la base de datos con cada palabra seleccionada. De cada palabra obtenemos un número de imágenes predeterminado. Después se compara cada imagen obtenida de cada palabra buscada con la imagen que queremos etiquetar. Por último se escoge la palabra cuyas imágenes hayan obtenido los mejores resultados.

Esta sería el funcionamiento general del algoritmo, a continuación entraremos a describir cada parte en mayor detalle.

2.1. Selección de las palabras más relevantes del texto.

El primer paso de nuestro algoritmo es seleccionar del texto las palabras más relevantes. Para ello hay que leer todas las palabras e ir filtrándolas hasta obtener el número de palabras que deseemos, las más importantes.

El primer paso es eliminar las palabras más comunes del lenguaje así como preposiciones, artículos, conectores, etc. Eliminamos las palabras que se encuentren recogidas en un archivo de palabras comunes, que denominaremos “Longlist”.

Ahora debemos fijarnos en que muchas de las palabras que tenemos pueden ser conjugaciones de un mismo verbo, plurales, cambio de género,… En definitiva proceden de la misma raíz (o en inglés *stem*). **Stemming** [2] es un método para reducir una palabra a su raíz. Hay algunos algoritmos de stemming que ayudan en sistemas de recuperación de información. Stemming aumenta el *recall*, que es una medida sobre el número de documentos que se pueden encontrar con una consulta. Por ejemplo una consulta sobre "bibliotecas" también encuentra documentos en los que solo aparezca "bibliotecario" porque el stem de las dos palabras es el mismo ("bibliotec").

El algoritmo más común para stemming, y que utilizaremos en nuestra aplicación, es el algoritmo de Porter [3]. El algoritmo de Stemming de Porter (o 'Porter stemmer') es un procedimiento para eliminar los terminaciones morfológicas y flexivas más comunes de las palabras en inglés. Su uso principal es como parte de un proceso de normalización de términos que se suele hacer en la creación de sistemas de recuperación de información (*information retrieval* [1]).

Entonces primeramente extraemos el stem de cada palabra y agruparemos las palabras en grupos con el mismo stem, para quedarnos luego con solo una palabra por stem, la palabra más corta. Por ejemplo: si tenemos las palabras “teach”, “teaching” y “teacher”, las

tres tienen el mismo stem, “teach”, nos quedaríamos con la palabra “teach”. También contamos las veces que aparece en el texto (en el ejemplo serian 3 veces).

Muy bien, ahora tenemos una lista de palabras y las veces que aparecen en el texto. A continuación necesitamos ordenarlas de alguna forma por su relevancia. Para saber la importancia de una palabra primero tomamos una medida de lo común que es. Por ejemplo, “Person” es más común que “Obama”, siendo más relevante para nosotros la segunda.

Para evaluar la importancia, disponemos de un archivo con todas las palabras que aparecen en la versión corta del British National Corpus [4] [5] con el número de veces que aparece. Denominaremos este archivo como “ShortCorpus”. Entonces buscamos cada palabra y calculamos su peso como la frecuencia que aparece (nº de veces que aparece / palabras totales) por el número de veces que la palabra aparece en nuestro texto.

Con esto asignamos a cada palabra un peso que evalúa su nivel de relevancia, y escogemos las N palabras con mayor peso.

2.2. Búsqueda de imágenes.

Como hemos visto en el diagrama general, una vez que tenemos las palabras más relevantes, realizamos búsquedas sobre una base de datos de imágenes. Para cada palabra busca un número de imágenes que le indiquemos en la base de datos de imágenes y las almacena para que sean comparadas más adelante con nuestra imagen.

2.3. Comparación de imágenes.

La comparación de dos imágenes se puede realizar de muchas maneras distintas. Nuestro algoritmo comparará las dos imágenes siguiendo un proceso de dos pasos: comparación de histograma de bloques (blok-histogram comparison) y comparación de bordes (edge comparison).

- **Block-histogram comparison:**

  - **Calculo del histograma:**

    En estadística, un histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados, ya sea en forma diferencial o acumulada. [6]

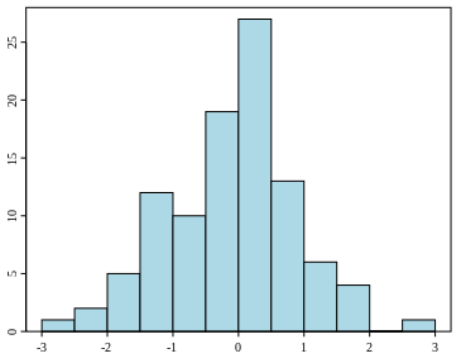

Figura 2: Histograma

En el caso de las imágenes, lo que realizaremos es un histograma de color. Los histogramas de color son estructuras que pueden ser creadas a partir de imágenes de diferentes espacios de color, o sea RGB, cromaticidad RG o cualquier espacio de color en cualquier dimensión. Un histograma de una imagen es producido primero a partir de la discretización de los colores en una imagen en números de grupos, y luego se cuenta el número de pixeles de la imagen en cada grupo. [7]

Por tanto, el histograma es una colección de datos sobre el color, organizados en unos bloques predefinidos. A nosotros nos interesa la intensidad del color, pero los datos recogidos pueden ser cualquier característica que describa la imagen.

Veamos un ejemplo sencillo. [8] Imagine que una matriz contiene información de una imagen (por ejemplo la intensidad en el rango 0-255):

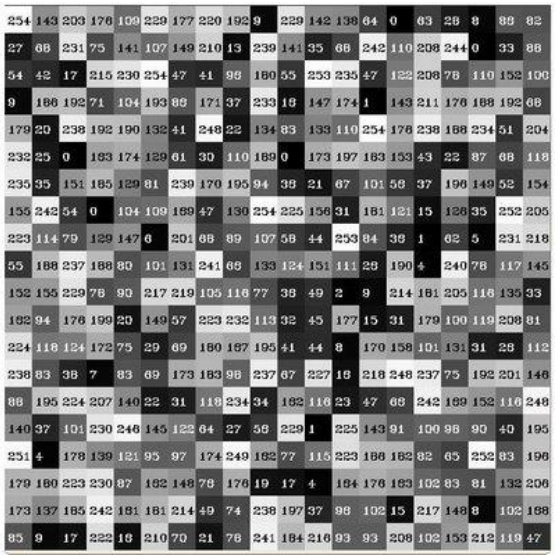

Figura 3: Matriz con información imagen

¿Qué pasa si queremos contar estos datos de manera organizada? Ya que sabemos que el rango de valor de la información para este caso es de 256 valores, podemos segmentar nuestra gama en las subpartes (llamados *bin*) como:

$$[0, 255] = [0, 15] \cup [16, 31] \cup .... \cup [240, 255]$$
$$\text{range} = \text{bin}_1 \cup \text{bin}_2 \cup .... \cup \text{bin}_{n=15}$$

Figura 4: Segmentos matriz imagen

Y podemos llevar la cuenta del número de píxeles que se encuentran en el rango de cada bin_i. Aplicando esto al ejemplo anterior se obtiene la imagen de abajo (eje x representa los *bin* y el eje y el número de píxeles en cada una de ellas).

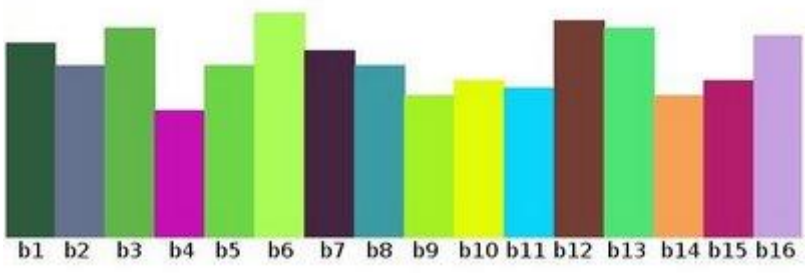

Figura 5: Resultado ejemplo histograma

Esto fue sólo un ejemplo sencillo de cómo funciona un histograma y por qué es útil. Vamos a identificar algunas partes del histograma:

- dims: El número de parámetros que desea recopilar de los datos. En nuestro ejemplo, dims = 1 porque sólo estamos contando los valores de intensidad de cada píxel.
- bins: Es el número de subdivisiones en cada dim. En nuestro ejemplo, bins = 16.
- rango: Los límites para los valores que se deben medir. En este caso: rango = [0,255].

Una vez explicado el concepto con un ejemplo sencillo, pasemos a ver como calcular el histograma de una imagen en nuestro algoritmo.

En primer lugar se divide la imagen en sus canales R (rojo=red), G (verde=green), B (azul=blue). Luego se calcula el histograma para cada canal:

- Igual que en el ejemplo sencillo, dims = 1 porque sólo estamos contando los valores de intensidad de cada píxel.
- Dado que estamos trabajando en los canales R, G y B, sabemos que nuestros valores variarán en el intervalo [0,255].
- Usaremos 256 como número de *bins.* Así tenemos 256 divisiones (de 0 a 255).

Con esto obtenemos 3 matrices con los datos de intensidad para cada canal RGB. Si representáramos esos datos en una gráfica nos quedaría algo como esto:

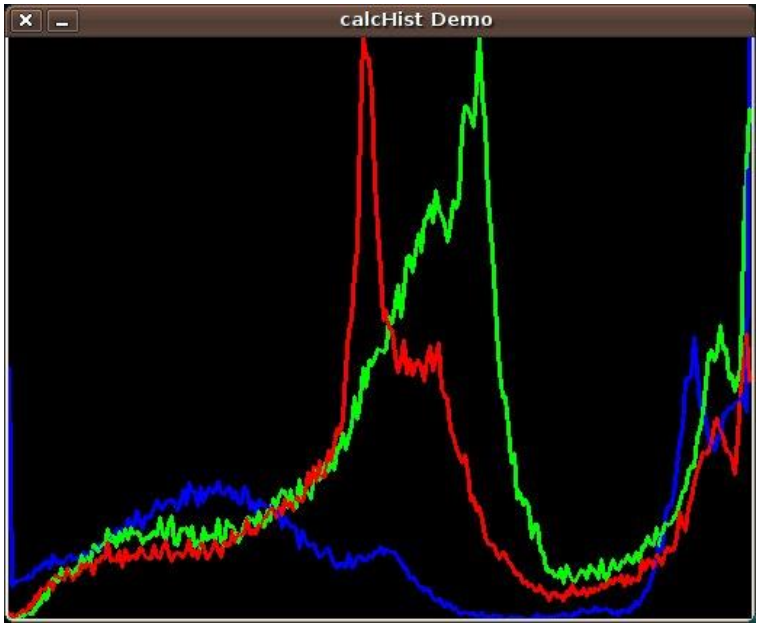

Figura 6: Histograma de color

- **Comparación de los histogramas:**

  Ya sabemos cómo calculamos el histograma de una imagen. Ahora vamos a ver el proceso que sigue el algoritmo para comparar dos imágenes por su histograma.

  1- Divide cada imagen en N bloques iguales.
  2- Para cada bloque calcula su histograma en los canales RGB como vimos anteriormente y calcula la media y desviación en cada canal. Obtenemos así para cada bloque la media y desviación de la intensidad en los canales RGB.
  3- Cuando ya tenemos los datos de las dos imágenes, las comparamos calculando la suma de las diferencias de las dos imágenes (distancia entre ellas):
     a. Primero sumamos las diferencias en los canales R, G, y B.
     b. La suma total es la suma de los valores de los tres canales calculados antes.

➢ **Edge comparison:**

- **Algoritmo de Canny:**

  La comparación de dos imágenes por bordes se basa en el algoritmo de Canny [9]. Algoritmo de Canny es un operador desarrollado por John F. Canny en 1986 que utiliza un algoritmo de múltiples etapas para detectar una amplia gama de bordes en imágenes.
  El propósito de Canny era descubrir el algoritmo óptimo de detección de bordes. Para que un detector de bordes pueda ser considerado óptimo debe cumplir los siguientes puntos:

  - Buena detección: el algoritmo debe marcar el mayor número real en los bordes de la imagen como sea posible.
  - Buena localización: los bordes de marca deben estar lo más cerca posible del borde de la imagen real.

- Respuesta mínima: El borde de una imagen sólo debe ser marcado una vez, y siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Pasos [10]:

1- Filtrar el ruido. El filtro de Gauss se usa para este propósito. He aquí un ejemplo de un filtro gaussiano 5x5:

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Figura 7: Filtro Gaussiano

2- Encontrar el gradiente de intensidad de la imagen. Para ello, seguimos un procedimiento análogo a Sobel [11]. El operador Sobel calcula el gradiente de la intensidad de una imagen en cada punto (píxel):

   a. Aplicar un par de máscaras de convolución [12]:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Figura 8: Máscaras de convolución

   b. Encontrar la fuerza de gradiente y dirección con:

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Figura 9: Gradiante y dirección

3- Se aplica no supresión máxima. Esto elimina los píxeles que no se consideran parte de un borde. Por lo tanto, sólo las líneas finas (bordes candidatos) se mantendrán.
4- Histéresis: El paso final. Canny hace uso de dos umbrales (superior e inferior):
    a. Si un gradiente del píxel es mayor que el umbral superior, el píxel se acepta como un borde.
    b. Si un valor de gradiente del píxel está por debajo del umbral inferior, entonces se rechaza.
    c. Si el gradiente del píxel está entre los dos umbrales, entonces será aceptado sólo si está conectado a un píxel que está por encima del umbral superior.

Veamos el resultado de aplicar el algoritmo a una imagen:

Figura 10: Ejemplo algoritmo de Canny

- **Comparación de los bordes:**

Veamos ahora como realiza nuestro algoritmo la comparación de dos imágenes. Se aplica el algoritmo de Canny a las dos imágenes y obtenemos sus correspondientes resultados como vimos en el ejemplo anterior. A continuación aplicamos la

comparación por histogramas visto anteriormente en este capítulo, solo que ahora se realiza en un solo canal ya que las imágenes son en blanco y negro.

- **Resultado final:**

  El resultado final es la suma de las comparaciones por histograma y por bordes. Con esto obtenemos el valor de la diferencia que hay entre las dos imágenes, cuanto menor es el resultado, más parecidas son las imágenes, siendo 0 si las dos son la misma imagen.

2.4. Etiquetado de imagen.

El proceso principal conecta los distintos procesos intermedios y decide con qué palabra se etiquetará la imagen.

Una vez obtenidos los resultados de comparar todas las imágenes obtenidas con nuestra imagen, se aplica un algoritmo para decidir que palabra obtiene los mejores resultados.

Este algoritmo puede ser simplemente la media aritmética de los resultados de cada palabra, la palabra que haya obtenido el mejor resultado, o cualquier otro algoritmo más desarrollado.

3. TECNOLOGÍAS EMPLEADAS.

En este capítulo expondremos las distintas tecnologías y herramientas utilizadas para desarrollar el proyecto.

3.1. Base de datos de imágenes: Bing.

Como hemos visto en el capítulo anterior, necesitamos buscar y obtener imágenes para compararla con la imagen que queremos asignar la etiqueta. Necesitamos una base de datos lo más extensa posible. Actualmente hay potentes motores de búsqueda como Google Images o Bing que proporcionan este servicio.

La primera opción que barajamos, fue Google Images. El primer problema que encontramos fue que la antigua API de búsqueda está obsoleta, siendo sustituida por una nueva API: Custom Search API [13]. Investigando sobre este servicio, nos encontramos con que el límite de consultas diarias es limitado. Además no encontramos demasiada información sobre el uso de la librería o ejemplos prácticos.

La siguiente opción fue investigar el servicio de Microsoft, Bing [14]. Las características del servicio, el número de transacciones gratuitas que ofrece y la documentación que proporciona nos hicieron decidir por esta opción.

- **Bing Search API:**

La API de Bing Search permite a los desarrolladores integrar los resultados de búsqueda en aplicaciones o sitios web utilizando XML o JSON. Añadir funcionalidad de búsqueda a un sitio web, crear aplicaciones de consumo o empresariales únicas.

La API de búsqueda de Bing ofrece múltiples tipos de fuentes (o tipos de resultados de la búsqueda). Puede solicitar un único tipo de fuente o diversos tipos de fuentes con cada consulta. Por ejemplo, puede solicitar los resultados web, imágenes, noticias y vídeo para una sola consulta de búsqueda.

Bing Search API devuelve resultados para los siguientes tipos de fuentes:

Tipo de fuentes	Descripción
Web	Resultados de la búsqueda web
Imágenes	Resultados de la búsqueda de imágenes
Noticias	Resultados de la búsqueda de noticias
Videos	Resultados de la búsqueda de vídeo
Búsquedas Relacionadas	Buscar sugerencias relacionadas basadas en la consulta introducida
Sugerencias de ortografía	Sugerencias de ortografía en base a la consulta introducida

Tabla 1: Tipo de fuentes de búsqueda

Nosotros utilizaremos los resultados de búsqueda de imágenes para nuestro proyecto.

Primero tenemos que registrarnos y contratar el servicio. El servicio básico ofrece 5000 transacciones al mes gratuitamente. Si necesitamos más debemos contratar alguno de las opciones disponibles a distintos precios.

Una vez contratado el servicio, se pone a nuestra disposición para descargarnos la biblioteca de clases para que la incluyamos en nuestro proyecto. Al ser un servicio de Microsoft, la biblioteca viene implementada en C# para su uso en plataformas .Net o programas de Windows. Esto condicionará la decisión de desarrollo del proyecto como veremos más adelante cuando entremos a explicar el lenguaje utilizado.

3.2. Lenguaje de programación y entorno de desarrollo.

La primera opción que pensábamos al inicio del proyecto era realizar la aplicación en Java usando Eclipse como entorno de desarrollo, pero como hemos visto, al elegir Bing como motor de búsqueda, la biblioteca proporcionada está en C#. Por tanto al final se optó por desarrollar el proyecto en C# con entorno de desarrollo Microsoft Visual Studio 2013.

- **Lenguaje C# (C Sharp):**

C# [15] Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

- **Microsoft Visual Studio 2013:**

Microsoft Visual Studio [16]es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, como Java, C++, C o el que nos ocupa, C#.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc.

En nuestro caso utilizaremos la versión de 2013 para desarrollar los distintos módulos y una aplicación de consola en Windows para probar los resultados.

3.3. Comparación de imágenes: OpenCV

Para la comparación de imágenes usaremos una biblioteca llamada OpenCV [17]. OpenCV (Open Source Computer Vision Library) es una biblioteca libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos.

La biblioteca incluye muchísimas áreas, pero nosotros utilizaremos los kits de funciones de tratamiento de imágenes que dispone para obtener el histograma y el algoritmo de Canny para comparación por bordes.

4. DISEÑO.

4.1. Arquitectura de la aplicación.

La aplicación está dividida en cuatro módulos independientes que implementan el algoritmo y un programa principal que lo ejecuta. Cada módulo implementa una de los procesos del algoritmo expuesto en al apartado 2 de este documento. Cada uno esta implementado en paquetes independientes para poder ser exportados y utilizados en otros proyectos.

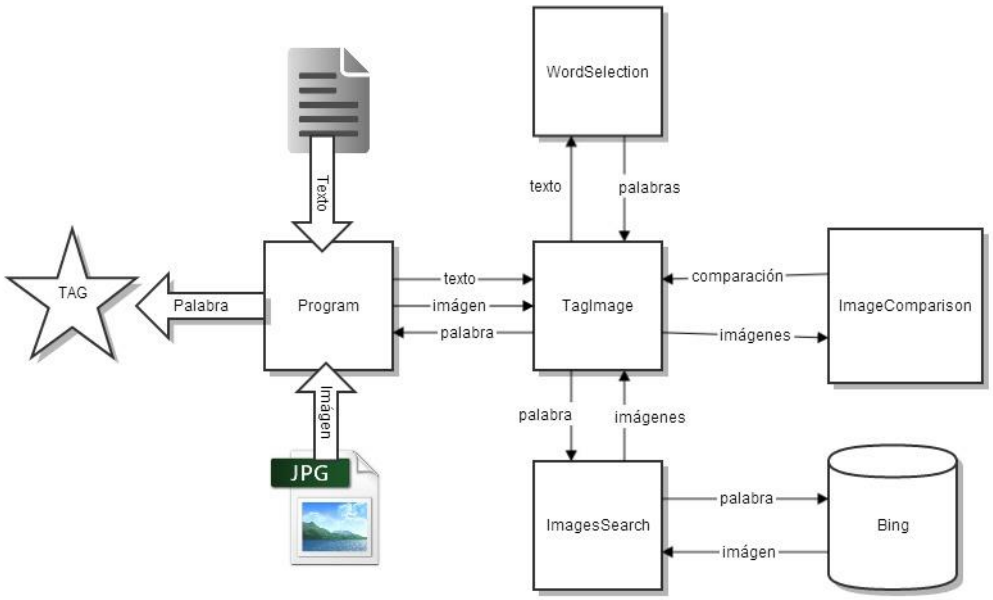

Figura 11: Arquitectura de la aplicación

4.2. WordSelection.

Este módulo implementa la primera parte del algoritmo, la selección de las palabras más relevantes del texto.

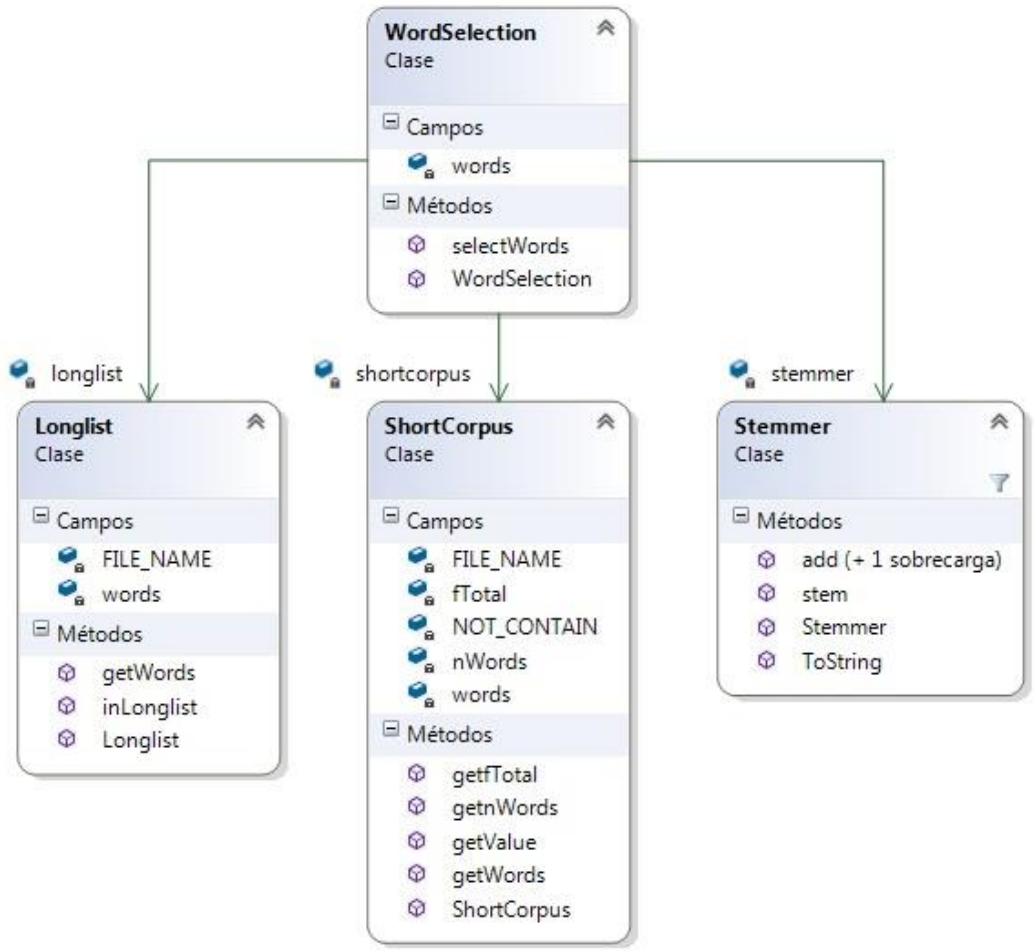

Figura 12: Diagrama de clases WordSelection

Consta de cuatro clases:

- **Longlist**:

  Esta clase implementa el filtro de palabras comunes.

  - **Constructor**: Accede al fichero "Longlist.txt" y guarda las palabras en la lista ***words***.

  - **inLonglist:** Comprueba si una palabra está en la lista. Recibe una palabra y devuelve true o false si la palabra está o no en la lista.

  - **getWords:** Devuelve la lista de palabras.

- **ShortCorpus**:

Esta clase implementa el cálculo de la frecuencia de ocurrencia de una palabra en el British National Corpus [4] [5], utilizado para valorar su importancia.

- **Constructor:** En esta clase se accede al fichero "ShortCorpus.txt ", que vimos en el apartado 2. El fichero tiene este formato:

```
# smallish English corpus.
#
54599      89740556
#
5776384 the
2789403 of
2421302 and
1939617 a
……..
```

Figura 13: Formato ShorCorpus.txt

Primero encontramos el número total de entradas en el archivo, ***nWords*** (54599), acompañado del número total de palabras en el corpus, ***fTotal*** (89740556). Luego encontramos cada palabra precedida del número de ocurrencias en el corpus. Para cada palabra calcula su frecuencia como el número de veces que aparece entre el número total de palabras (por ejemplo, la frecuencia de la palabra "the" será 5776384 / 89740556). Ajustamos el resultado calculando $\log 1/f$. El resultado se guarda en un lista, ***words***, con clave la palabra y valor el resultado obtenido para su frecuencia.

- **getWords:** Devuelve una lista con las palabras y su frecuencia.

- **getValue:** Devuelve la frecuencia de una palabra. Si la palabra no se encuentra en el corpus, se realiza el cálculo de su frecuencia como si el número de veces que aparece es 1.

- **getnWords:** Devuelve el número de entradas en el archivo.
- **getfTotal:** Devuelve el número total de palabras en el corpus.

- **Stemmer:**

  Esta clase implementa el Algoritmo de Stemming de Porter. Podemos obtener el algoritmo implementado en múltiples lenguajes desde la propia página del creador del algoritmo, Martin Porter [3]. En nuestro caso descargamos la versión en C#. [18]

  - **add:** Recibe una palabra en forma de array de caracteres y su longitud. Añade los caracteres de la palabra al buffer para hacer el stemming.

  - **stem:** Realiza el stemming de la palabra.

  - **ToString:** Devuelve el stem producido en forma de string.

- **WordSelection:**

  Esta clase implementa la selección de las palabras más importantes del texto. Para ello utiliza las clases anteriores y sus métodos.

  - **selectWords:** Recibe la ruta del archivo de texto del que se desea extraer las palabras más importantes y el número de palabras que queremos. Devuelve una lista con las palabras obtenidas con sus pesos. El método sigue los siguientes pasos:
    1- Obtiene todas las palabras del texto y las guarda en una lista.
    2- Elimina las palabras incluidas en el shortlist.
    3- Realiza el stemming de cada palabra. Agrupa todas las palabras que producen el mismo stem. Guarda cada stem con la lista de palabras que la producen en otra lista.
    4- Por cada stem producido elige la palabra más corta y cuenta cuantas veces aparece. Guarda la palabra y el número en una lista.
    5- A cada palabra se le asigna un peso que corresponde con su importancia. Este peso se calcula como el número de veces que aparece en el texto multiplicado por la frecuencia obtenida del ShortCorpus.
    6- Normaliza los pesos.
    7- Ordena la lista de mayor a menos peso de las palabras

8- Devuelve una lista con las N palabras solicitadas con sus pesos.

4.3. ImagesSearch.

Este módulo implementa la búsqueda y obtención de imágenes en la base de datos. Como vimos en el apartado 3, vamos a utilizar Bing como motor de búsqueda.

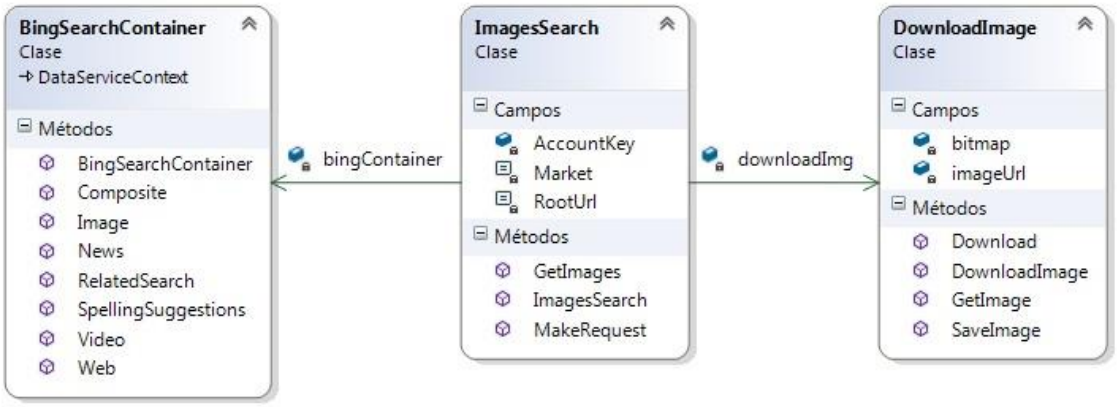

Figura 14: Diagrama de clases ImagesSearch

Consta de tres clases:

- **BingSearchContainer:**

  Biblioteca de clases de la API de búsqueda de Bing. Obtenida una vez nos hemos suscrito al servicio, esta biblioteca incluye todos los métodos necesarios para realizar las búsquedas. En nuestro caso sólo vamos a utilizar los métodos de búsqueda de imágenes pero incluye también búsquedas web, videos o noticias.

  - **Constructor:** El constructor de la clase recibe la URI del servicio de búsqueda de Bing al que estemos suscritos:

Tipo de suscripción	URI del servicio
Bing Search API	https://api.datamarket.azure.com/Bing/Search/
Bing Search API - Web Results Only	https://api.datamarket.azure.com/Bing/SearchWeb/

Tabla 2: Tipos de suscripción Bing

En nuestro caso utilizaremos la primera puesto que buscamos imágenes.

- **Image:** Construye la consulta para el servicio de búsqueda de imágenes. Los parámetros que recibe son:

Nombre	Tipo	Ejemplo	Descripción
Query	String	Obama	Palabra que se desea buscar.
Adult	String	Moderate	Configuración que se utiliza para el filtrado de contenido sexualmente explícito. Si no se especifica el nivel de filtrado significa que la API utilizará el nivel predeterminado para un mercado en particular, que puede variar según el mercado. Ver Adult
Latitude	Double	47.603450	Latitud. Rango desde –90 a 90.
Longitude	Double	-122.329696	Longitud. Rango desde –180 a 180.
Market	String	en-US	Mercado. Si no se especifica el parámetro, la API intenta determinar un mercado aplicable a través del uso de la lógica tal como la dirección IP de la solicitud, cookies, y otros elementos. Ver Mercado
Options	String	EnableHighlighting	Especifica opciones generales para la consulta. Ver Opciones.
ImageFilters	String	Size:Small	Array de strings que filtran la respuesta que la API envía basado en el tamaño, aspecto, color, estilo, o cualquier combinación de los mismos. Ver Filtros

Tabla 3: Parametros Image

El único parámetro obligatorio es *Query,* los demás son opcionales.

Opciones:

Valores del parámetro *Options* de la consulta.

Valor	Descripción
DisableLocationDetection	Previene que Bing infiera la localización a partir del término de la consulta
EnableHighlighting	Bing utilizará un carácter especial para identificar el principio y el final de un término de consulta que aparece en los resultados. [19]

Tabla 4: Valores del parámetro *Options*

Puede especificar múltiples opciones concatenando las opciones con el signo más (+). Ejemplo:

Options='EnableHighlighting+DisableLocationDetection'

Adult:

Valores del parámetro *Adult* especifican el nivel del filtro de contenido adulto de la consulta.

Valor	Descripción
Off	No se filtra la consulta.
Moderate	Los resultados de una consulta no deben incluir imágenes o videos de sexo explícito, pero pueden incluir texto sexualmente explícito.
Strict	Los resultados de una consulta no deben incluir texto sexualmente explícito, imágenes o videos.

Tabla 5: Valores del parámetro *Adult*

ImageFilters:

Valores del parámetro *ImageFilters* restringen los resultados.

Valor	Restringir los resultados de imágenes por
Size:Small	Tamaño pequeño.
Size:Medium	Tamaño mediano.
Size:Large	Tamaño grande.
Size:Height:<Height>	Altura en pixels, donde <Height> es un valor entero sin signo
Size:Width:<Width>	Anchura en pixels, donde <Width> es un valor entero sin signo
Aspect:Square	Standard aspect ratio.
Aspect:Wide	Widescreen aspect ratio.
Aspect:Tall	Tall aspect ratio.
Color:Color	Imágenes en Color.
Color:Monochrome	Imágenes blanco y negro
Style:Photo	Fotos.
Style:Graphics	Gráficos o ilustraciones
Face:Face	Caras.
Face:Portrait	Retrato (cabeza y hombros).
Face:Other	Otros.

Tabla 6: Valore del parámetro ImageFilters

Mercado:

Valores del parámetro Market indican el idioma y el país/región. Como la lista es muy grande incluiré, a modo de ejemplo, el de España y Estados Unidos.

Market ID	Idioma	País/Región
en-US	Inglés	Estados Unidos
es-ES	Español	España

Tabla 7: Valores del parámetro Market

- **Añadir opciones a la consulta (AddQueryOption)**

  El método anterior devuelve la consulta que queremos realizar. Tras esta consulta podemos añadir otras opciones como el número de resultados que deseamos obtener:

Parámetro	Descripción
$top	Especifica el número de resultados a devolver. El valor predeterminado es 50
$skip	Especifica el desplazamiento para los resultados. El valor predeterminado es cero.
$format	Especifica el formato de la respuesta. La opciones son Atom (para XML) o JSON. Predeterinado: Atom

Tabla 8: Opciones de la consulta

Ejemplo: imageQuery.AddQueryOption("$top", 25) para devolver 25 resultados

- **Resultado de ejecutar la consulta (Execute):**

  Una vez preparada la consulta, la ejecutamos. El resultado de realizar la consulta es un objeto de la clase *ImageResult* con los siguientes atributos:

Name	Type	Description
ID	Guid	Identificador
Title	String	Título de la imagen
MediaUrl	String	URL de la imagen a tamaño completo
SourceUrl	String	URL de la web que contiene la imagen
DisplayUrl	String	URL de la página de resultados de la búsqueda
Width	Int32	Ancho de la imagen en tamaño completo, en píxeles, si está disponible
Height	Int32	Altura de la imagen en tamaño completo, en píxeles, si está disponible
FileSize	Int64	Tamaño del archivo de imagen en tamaño completo, en bytes, si está disponible
ContentType	String	Tipo MIME de la imagen, si está disponible
Thumbnail	Thumbnail	Propiedades de la miniatura de la imagen

Tabla 9: Resultado de ejecutar la consulta

- **DownloadImage:**

Esta clase implementa los métodos para descargar y guardar una imagen de una URL web.

- **Constructor:** Recibe la URL de la imagen.

- **Download:** Obtiene la imagen a partir de la URL. La imagen se guarda en una variable Bitmap. Bitmap es un objeto que se utiliza para trabajar con imágenes definidas mediante datos de píxeles. [20]

- **SaveImage:** Recibe el nombre con que queremos guardar la imagen y el formato (JPEG, GIF, etc). Guarda la imagen con el formato indicado en el disco.

- **ImagesSearch:**

Esta clase implementa los métodos para buscar las imágenes, usando las clases anteriores.

- **Constructor:** Recibe la clave de la cuenta de Bing y crea el contenedor de Bing.
- **MakeRequest:** Recibe la palabra que queremos buscar y el número de resultados que queremos obtener. El método crea la consulta y la ejecuta. Devuelve una lista con las URL de las imágenes obtenidas.

- **GetImages:** Recibe la palabra que queremos buscar y el número de resultados que queremos obtener. El método primero ejecuta MakeRequest, obteniendo así una lista con las URL de las imágenes. A continuación descarga y guarda las imágenes en una carpeta “downloads” con el nombre de imagen <palabra><N>.<formato> siendo “palabra” la palabra que hemos buscado”, “N” el número de resultado y “formato” JPEG. Por ejemplo, si buscamos “Obama” y obtenemos 3 resultados, se guardará: Obama1.jpg, Obama2.jpg y Obama3.jpg. Se guardan las rutas a las imágenes en una lista. Se retorna la lista con las rutas a las imágenes obtenidas.

4.4. ImagesComparison.

Este módulo implementa la comparación de dos imágenes. Para ello se utilizan las clases y métodos de la librería OpenCV vista en el capítulo 3.

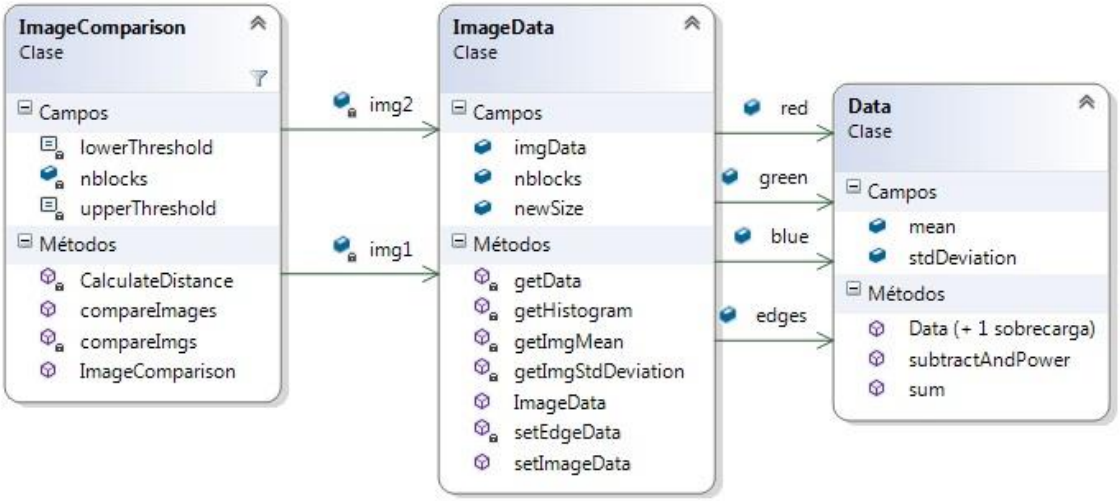

Figura 15: Diagrama de clases ImagesComparison

Se compone de una clase principal (ImageComparison) y dos clases auxiliares para el manejo de los datos.

- **ImageComparison:**

Es la clase principal. Incluye los métodos para comparar dos imágenes.

- **Constructor:** Recibe el número de bloques en que se van a dividir las imágenes.

- **compareImages:** Recibe el nombre de dos imágenes. Y devuelve la distancia entre ellas, valor de comparar las dos imágenes. Realiza los siguientes pasos:
    1. Carga las dos imágenes. Utiliza el método imRead [21] de OpenCV para cargar la imagen y devolver los datos en una matriz Mat.
    2. Inicializa los dos atributos ImageData para las dos imágenes cargadas.
    3. Calcula los datos de las dos imágenes: Divide las imágenes en bloques de NxN y calcula la media y desviación de cada una de ellas, aplicando el método setImageData.

4. Devuelve el resultado de la comparación entre las dos imágenes, aplicando el método compareImgs.

- **compareImgs:** Recibe los dos objetos ImageData correspondiente a las dos imágenes y el número de bloques en que se dividen. El método compara las dos imágenes usando los dos métodos explicados en capítulo 2, la comparación de histogramas y la comparación de bordes. Sigue los siguientes pasos:

    1. Para cada bloque:
        1. Calcula la diferencia entre las medias y desviaciones de las dos imágenes para los tres canales R, G y B, y de los bordes, aplicando el método subtractAndPower.
        2. Suma el resultado al total para cada uno (R, G, B y bordes).
    2. El resultado final de la comparación por histograma es la suma de la media y la desviación de los tres canales R, G, B.
    3. El resultado final de la comparación por bordes es la media y desviación calculada en el paso 1 porque sólo hay un canal.
    4. Devuelve la distancia de las dos imágenes aplicando el método CalculateDistance.

- **CalculateDistance:** Recibe los datos calculados por los dos métodos, histograma y bordes. Devuelve la distancia calculada a partir de los datos obtenidos por los dos métodos. El resultado es la suma de la media con la desviación.

- **<u>ImageData:</u>**

  En esta clase se implementan los métodos para calcular los histogramas e implementar el algoritmo de Canny para la comparación por bordes.

  - **Constructor:** Recibe la matriz Mat de la imagen y el número de bloques en que se va a dividir. Inicializa los atributos red, green, blue y edge como matriz de Data de NxN bloques. Redimensiona la imagen a 512x512 pixeles para que las imágenes tengan la misma dimensión al compararlas, utilizado el método resize [22] de

OpenCV. El resultado lo guarda en imgData. Esta matriz es la que se usará para los cálculos.

- **setImageData**: Divide la imagen en N bloques de igual tamaño, obtiene el histograma de los tres canales de color (RGB) de cada bloque. Obtiene la media y el valor de desviación típica de cada bloque. Para partir la imagen los tres canales RGB se utiliza el método Split [23] de OpenCV. Para calcular el histograma se utiliza el método getHistogram.

- **setEdgeData:** Aplica el algoritmo de Canny a la imagen utilizando el método Canny [24] de OpenCV. Divide la imagen obtenida en N bloques de igual tamaño, obtiene el histograma de cada bloque utilizando el método getHistogram. Obtiene la media y el valor de desviación típica de cada bloque.

- **getHistogram:** Obtiene el histograma de una imagen. Utiliza el método calcHist [25] de OpenCV.

- **getData:** Obtiene la media y desviación de una imagen.

- **getImgMean:** Calcula la media de una imagen.

- **getImgStdDeviation:** Calcula la desviación de una imagen.

- **Data:**

En esta clase se guardan la media y desviación de una imagen.

- **Constructor:** Recibe el valor de la media y de la desviación de la imagen.

- **subtractAndPower:** Recibe los valores la imagen con la que se quiere hacer la resta. Realiza el cuadrado de la diferencia para la media y para la desviación. Devuelve los valores en un nuevo objeto Data.

- **sum:** Recibe un objeto Data y suma sus valores de media y desviación a los de la imagen.

4.5. TagImages.

Este es el módulo principal de la aplicación. En él se implementa la funcionalidad de elegir la mejor palabra para etiquetar una imagen. Para ello se comunica con los otros módulos como vemos en la siguiente figura:

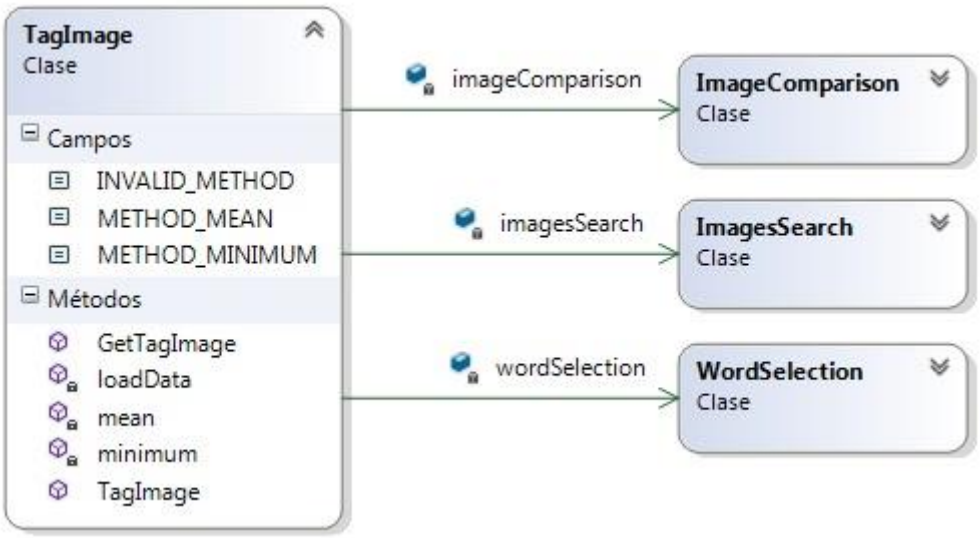

Figura 16: Diagrama de clases de TagImages

- **<u>TagImage</u>:**

  - **Constructor:** El constructor ejecuta el método LoadData.

  - **LoadData**: Carga los datos del fichero de configuración. En este fichero encontramos:
    - La clave de cuenta de Bing.
    - Número de bloques para la comparación de imágenes.
    - Número de palabras que seleccionar del texto.
    - Número de resultados que queremos obtener de la búsqueda.

- **GetTagImage:** Recibe el nombre del fichero de texto, el nombre de la imagen y la opción con el método utilizado para resolver que palabra es mejor. Los métodos implementados son:
    - METHOD_MEAN: media aritmética de todas las imágenes.
    - METHOD_MINIMUM: valor más bajo de todas las imágenes

  El método sigue los siguientes pasos:

1. Selecciona las palabras más importantes usando el método selectWords del objeto wordSelection.
2. Por cada palabra:
   2.1. Obtiene las imágenes usando el método GetImages del objeto imageSearch.
   2.2. Por cada imagen:
      2.2.1. Realiza la comparación de la imagen obtenida con la imagen que queremos etiquetar usando el método compareImages del objeto imageComparison. Guardando los resultados en una lista.
   2.3. Aplica el método indicado por parámetro sobre los resultados y lo guarda en una lista.
3. Elige la palabra que haya producido el mejor resultado.
4. Devuelve la palabra.

5. RESULTADOS.

En este capítulo mostraremos un ejemplo de ejecución del programa. Utilizaremos la imagen incluida en el Anexo a. y el texto incluido en el Anexo b. Con objeto de mostrar el proceso, mostraremos las salidas parciales del programa después de ejecutar cada módulo.

- **<u>Selección de palabras:</u>**

  - Seleccionando 10 palabras:

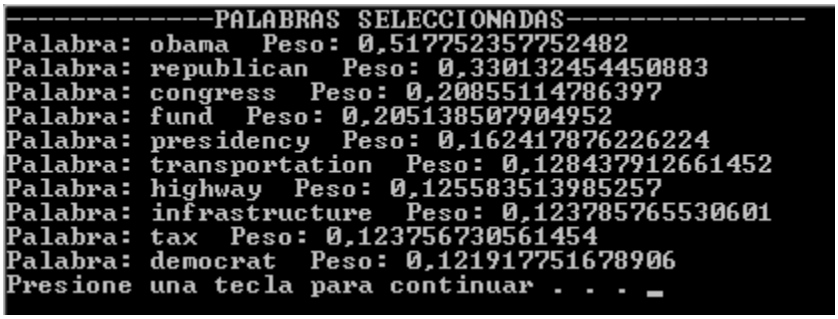

Figura 17: Seleccionando 10 palabras

- Seleccionando 4 palabras:

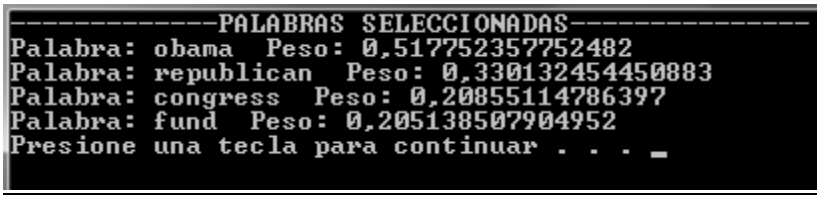

Figura 18: Selección de 4 palabras

- **Búsqueda de imágenes y comparación:**

El siguiente paso es buscar imágenes con cada palabra y comparar cada imagen obtenida con la imagen de entrada. Vamos a probar con el caso de 4 palabras. La búsqueda está configurada a 5 resultados. Se mostrará el resultado de la comparación de cada una de las cinco imágenes de cada palabra. Cuanto menor es el resultado, más parecidas son las imágenes, siendo el resultado 0 si son la misma imagen.

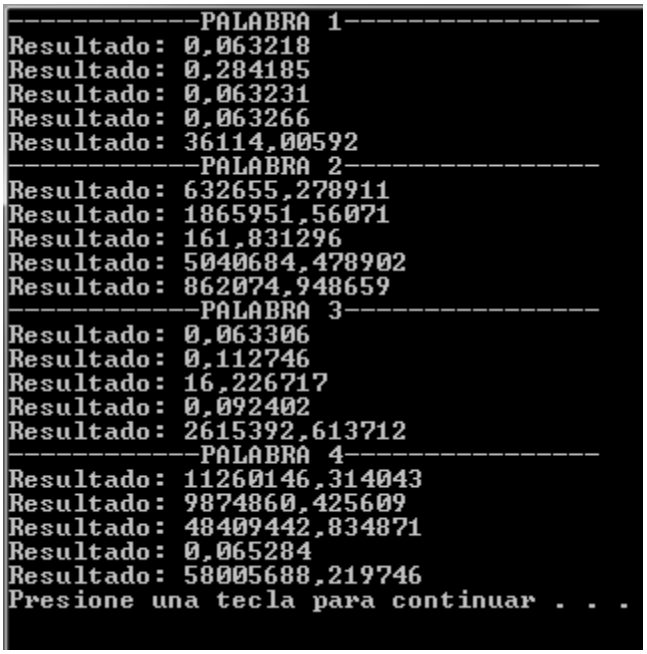

Figura 19: Resultado de comparar.

- **Selección de la etiqueta:**

El último paso es decidir que palabra obtiene el mejor resultado. Para eso hemos visto que disponemos de dos métodos: la media y el valor mínimo.

- Media:

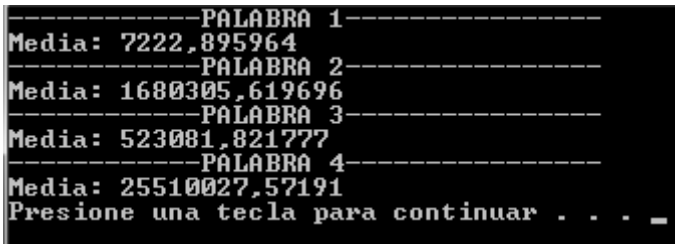

Figura 20: Media de las palabras.

- Mínimo:

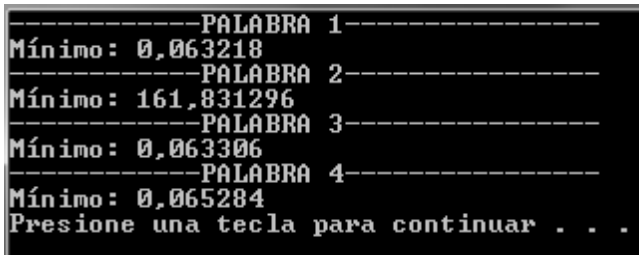

Figura 21: Valor mínimo de las palabras.

Vemos que el mejor resultado en ambos casos lo obtiene la primera palabra, “Obama”. Por tanto el programa nos mostrará esa palabra como resultado.

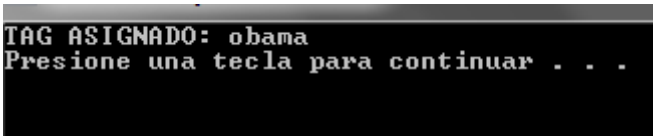

Figura 22: Tag Asignado.

6. CONCLUSIONES Y TRABAJO FUTURO.

En este proyecto hemos desarrollado un algoritmo para etiquetar una imagen usando una palabra que mejor la describa, partiendo de texto que habitualmente acompaña a la imagen.

Para empezar nos planteamos como íbamos a desarrollar la idea. Decidimos hacer que cada parte fuera un módulo separado para poder exportarlo como librería. En un principio nos planteamos hacerlo en Java, como se explica en el capítulo 3, de hecho ya teníamos hecho el módulo de selección de palabras. Cuando al final optamos por utilizar Bing y vimos que las librerías estaban en C#, decidimos cambiar de lenguaje.

El desarrollo de los módulos no presento demasiada complicación. Lo que si costó más fue integrar las librerías de OpenCV en nuestro proyecto.

A partir de aquí, el proyecto puede ir mejorando y varios aspectos. Se puede mejorar el algoritmo de selección de palabras, utilizando otras formas para decidir su importancia que no sean su frecuencia en el corpus. Podemos utilizar otras fuentes o desarrollar métodos más exhaustivos.

También podemos mejorar las búsquedas de imágenes jugando con los filtros y opciones que proporciona Bing. O podemos buscar otro motor de búsqueda.

En cuanto a la comparación de imágenes, hay muchas otras formas de realizar la comparación. Se precisaría un estudio más profundo sobre tratamiento de imágenes que escapan a mi conocimiento.

Por último se pueden proponer otros métodos para decidir qué resultados de comparación es mejor. Nosotros hemos propuesto dos muy sencillos, la media y el mínimo. Se podrían estudiar otros métodos o una combinación de varios.

En el futuro se podría utilizar esta aplicación para mejorar las búsquedas de imágenes en bases de datos, para proporcionar métodos de indexado de imágenes. Podemos utilizar la aplicación para extraer las imágenes de un libro o documento y guardarlas con un nombre que las describa, siendo útil por ejemplo para realizar búsquedas de imágenes en documentos.

De cualquier forma, esperamos poder continuar desarrollando y mejorando el proyecto en el futuro.

REFERENCIAS

[1] «WikiPedia - Information retrieval,» [En línea]. Available: http://en.wikipedia.org/wiki/Information_retrieval.

[2] «WikiPedia - Stemming,» [En línea]. Available: http://en.wikipedia.org/wiki/Stemming.

[3] «PorterStemmer,» [En línea]. Available: http://tartarus.org/martin/PorterStemmer/.

[4] «WikiPedia - British National Corpus,» [En línea]. Available: http://en.wikipedia.org/wiki/British_National_Corpus.

[5] «British National Corpus,» [En línea]. Available: http://www.natcorp.ox.ac.uk/.

[6] «Wikipedia - Histograma,» [En línea]. Available: http://es.wikipedia.org/wiki/Histograma.

[7] «WikiPedia - Histograma de color,» [En línea]. Available: http://es.wikipedia.org/wiki/Histograma_de_color.

[8] «OpenCV - Histogram Calculation,» [En línea]. Available: http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_calculation/histogram_calculation.html.

[9] «Wikipedia - Algoritmo de Canny,» [En línea]. Available: http://es.wikipedia.org/wiki/Algoritmo_de_Canny.

[10] «OpenCV - Canny Edge Detector,» [En línea]. Available: http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html.

[11] «WikiPedia - Operdador Sobel,» [En línea]. Available: http://es.wikipedia.org/wiki/Operador_Sobel.

[12] «Wikipedia - Convolución,» [En línea]. Available: http://es.wikipedia.org/wiki/Convoluci%C3%B3n.

[13] «Google Custom Search API,» [En línea]. Available:

https://developers.google.com/custom-search/json-api/v1/overview.

[14] «Bing Search API,» [En línea]. Available: https://datamarket.azure.com/dataset/bing/search.

[15] «Wikipedia - C#,» [En línea]. Available: http://es.wikipedia.org/wiki/C_Sharp.

[16] «WikiPedia - Microsoft Visual Studio.,» [En línea]. Available: http://es.wikipedia.org/wiki/Microsoft_Visual_Studio.

[17] «OpenCV,» [En línea]. Available: http://opencv.org/.

[18] «PorterStemmer - C#,» [En línea]. Available: http://tartarus.org/~martin/PorterStemmer/csharp.txt.

[19] «MSDN - Hit Highlighting,» [En línea]. Available: http://msdn.microsoft.com/en-us/library/gg447081.

[20] «MSDN - Bitmap (Clase),» [En línea]. Available: http://msdn.microsoft.com/es-es/library/system.drawing.bitmap.aspx.

[21] «OpenCV - imRead,» [En línea]. Available: http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html?highlight=imread#imread.

[22] «OpenCV - Resize,» [En línea]. Available: http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#resize.

[23] «OpenCV - Split,» [En línea]. Available: http://docs.opencv.org/modules/core/doc/operations_on_arrays.html?highlight=split#split.

[24] «OpenCV - Canny,» [En línea]. Available: http://docs.opencv.org/modules/imgproc/doc/feature_detection.html?highlight=canny#canny.

[25] «OpenCV - calcHist,» [En línea]. Available: http://docs.opencv.org/modules/imgproc/doc/histograms.html?highlight=calchist#calchist.

ANEXO A.

ANEXO B.

PRESIDENT OBAMA CALLED ON CONGRESSIONAL REPUBLICANS ON TUESDAY TO TAKE QUICK ACTION TO FUND INFRASTRUCTURE PROJECTS THROUGHOUT THE COUNTRY, ARGUING THAT FAILING TO DO SO COULD MEAN HUGE LAYOFFS for Americans this year.

Stepping up criticism of his opponents on Capitol Hill, Mr. Obama poked derisive fun at Republicans as he urged them to join Democrats to pass legislation that would replenish the Highway Trust Fund, which is expected to exhaust its resources by August.

I haven't heard a good reason why they haven't acted, Mr. Obama said in a speech at Georgetown Waterfront Park, overlooking the Potomac River and the Key Bridge, one of several bridges undergoing federally funded repairs after being deemed structurally deficient. It's not like they've been busy with other stuff. No, seriously!

The president said that if Congress did not act in the next couple of months, states would have to decide which projects to continue and which to halt, ultimately placing as many as 700,000 jobs at risk.

That would be like Congress threatening to lay off the entire population of Denver, or Seattle, or Boston, Mr. Obama told about 550 supporters and employees of the federal and Washington Transportation Departments. That's a lot of people. It would be a bad idea.

He made a pitch for his own infrastructure plan: a four-year, $302 billion initiative unveiled this year that would pay for renewing the transportation fund in part by closing corporate tax loopholes.

Mr. Obama said his plan was "sensible" and laughingly made reference to Republican criticisms of him. It's not crazy. It's not socialism. It's not the imperial presidency. No laws are broken, he said. We're just building roads and bridges.

And in keeping with his recent tactic of going around his Republican critics with executive action - which has prompted threats of a lawsuit by Republican congressional leaders - the president said he would not wait for Congress to act on infrastructure investments or a host of other priorities that he said they had neglected.

Middle-class families can't wait for Republicans in Congress to do stuff. So sue me, Mr. Obama said. As long as they're doing nothing, I'm not going to apologize for trying to do something.

Before a cabinet meeting on Tuesday morning, Mr. Obama said he was directing his team to look for creative ways to accomplish what Congress had refused to do.

If Congress can't act on core issues that would actually make a difference in helping middle-class families get ahead, then we're going to have to be creative about how we can make real progress, Mr. Obama said.

The transportation measure would fund projects in virtually every lawmaker's state or district and enjoys broad support, but it has been mired in a political fight over how to pay for it. Some Democrats favor increasing the 18.4-cent-per-gallon gas tax to bring in more revenue for highway programs, whose expenditures have outpaced the amount brought in by the tax. Republicans have resisted that approach, arguing that the measure should be paid for by cutting federal spending elsewhere.

Senator Ron Wyden, Democrat of Oregon and the chairman of the Senate Finance Committee, has proposed a package that would steer $8 billion to the Highway Trust Fund by closing tax loopholes. Representative Dave Camp, a Republican from

"Obama Urges Congress to Fund Infrastructure Projects" por JULIE HIRSCHFELD DAVIS

New York Times, JULY 1, 2014